Theory Comput Syst (2018) 62:337–348
https://doi.org/10.1007/s00224-017-9838-x

Shortest Augmenting Paths for Online Matchings on Trees

Bartłomiej Bosek[1] · Dariusz Leniowski[2] · Piotr Sankowski[2] · Anna Zych-Pawlewicz[2]

Published online: 24 January 2018

Abstract The shortest augmenting path (SAP) algorithm is one of the most classical approaches to the maximum matching and maximum flow problems, e.g., using it Edmonds and Karp (J. ACM **19**(2), 248–264 1972) have shown the first strongly polynomial time algorithm for the maximum flow problem. Quite astonishingly, although it has been studied for many years already, this approach is far from being fully understood. This is exemplified by the online bipartite matching problem. In this problem a bipartite graph $G = (W \uplus B, E)$ is being revealed online, i.e., in each round one vertex from B with its incident edges arrives. After arrival of this vertex we augment the current matching by using shortest augmenting path. It was conjectured by Chaudhuri et al. (INFOCOM'09) that the total length of all augmenting paths found by SAP is $\mathcal{O}(n \log n)$. However, no better bound than $\mathcal{O}(n^2)$ is known

The work of all authors was supported by Polish National Science Center grant 2013/11/D/ST6/03100. Additionally, the work of P. Sankowski was partially supported by the project TOTAL (No 677651) that has received funding from ERC.

✉ Bartłomiej Bosek
bosek@tcs.uj.edu.pl

Dariusz Leniowski
d.leniowski@mimuw.edu.pl

Piotr Sankowski
sank@mimuw.edu.pl

Anna Zych-Pawlewicz
anka@mimuw.edu.pl

[1] Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland

[2] Institute of Computer Science, University of Warsaw, Warsaw, Poland

even for trees. In this paper we prove an $\mathcal{O}(n \log^2 n)$ upper bound for the total length of augmenting paths for trees.

Keywords Online matchings · Bipartite matchings · Approximate matchings · Shortest augmenting paths · Dynamic graph algorithms

1 Introduction

The shortest augmenting path (SAP) algorithm is one of the most classical approaches to the maximum matching and maximum flow problems. Using this idea Edmonds and Karp in 1972 have shown the first strongly polynomial time algorithm for the maximum flow problem [5]. Quite astonishingly, although this idea is one of the most basic algorithmic techniques, it is far from being fully understood. It is easier to talk about it by introducing the online bipartite matching problem. In this problem a bipartite graph $G = (W \uplus B, E)$ is being revealed online, i.e., in each round one vertex from B with its incident edges arrives. After arrival of this vertex we augment this matching by using shortest augmenting path. It was conjectured by Chaudhuri et al. [4] that the total length of augmenting paths found by SAP is $\mathcal{O}(n \log n)$. However, no better bound than $\mathcal{O}(n^2)$ is known even for trees. Proving this conjecture would have quite striking consequences even for maximum flow problem, as it would show that the total length of augmenting paths in unit capacity networks in Edmonds-Karp algorithm is $\mathcal{O}(m \log n)$. This consequence is obtained via the bipartite line graph construction that is used to reduce the max-flow problem to maximum matching problem [10]. The obtained bipartite line graph has $2m$ vertices.

Our paper contributes to the study of SAP algorithm by showing that in the case of trees the total length of all augmenting paths is bounded by $\mathcal{O}(n \log^2 n)$. This result is obtained via the application of the heavy-light decomposition of trees [16] combined with charging technique that carefully assigns shortest augmenting paths to the structure of the tree. Although, this result seems to be restricted only to trees we be believe that it constitutes the first nontrivial progress towards resolving the above conjecture. Moreover, we actually conjecture here that trees are the worst-case examples for this problem. It seems that adding more edges can only help the SAP algorithm. In addition to that we explain why SAP is harder to analyze than other augmenting path algorithms, even though it seems way more natural.

2 Related Work

The online bipartite matching problem with augmentations has recently received increasing research attention [3, 4, 6, 7]. There are several reasons to study this problem. First of all, it provides a simple solution to the online bipartite matching algorithms used in many modern applications such as online advertising (e.g., Google Ads) [12] or client-server assignment [4]. Secondly, they could give rise to new effective offline bipartite matching algorithms as in [3]. Those new algorithms provide new insights to the old problem that was studied for decades.

In this paper we concentrate on bounding the total length of augmenting paths and not on the running time. With this respect, it was shown that if the vertices of B appear in a random order, the expected total paths' length for SAP is $\mathcal{O}(n \log n)$ [4]. The worst-case total length of paths remains an open question even for trees. In the class of trees the authors of [4] proposed a different augmenting path algorithm that achieves total paths' length of $\mathcal{O}(n \log n)$. On the other hand, for general bipartite graphs greedy ranking algorithm [3] guarantees $\mathcal{O}(n\sqrt{n})$ total length of paths.

First of all, the above study of online bipartite matching with augmentations should be related to the work of Gupta et al. [7] which shows an $\mathcal{O}(n)$ bound on the total length of paths, but allows to exceed the capacity of each server by a constant factor.

Another point of view is given by the dynamic matching algorithms. Most papers in this area consider edge updates in a general fully-dynamic model which allows for both insertions and deletions intermixed with each other. We note, however, that the exact results in this model [9, 15] do not imply any bound on the number of changes to the matching. Much faster update times can be achieved by constant approximate algorithms, for example [1, 14], which achieve polylogarithmic and logarithmic update times. Yet, the 2-approximation can be obtained in our setting by trivial greedy algorithm that preforms no changes at all.

Better approximation factor of $\frac{3}{2}$ was achieved by [13] in $\mathcal{O}(\sqrt{m})$ update time, and then improved by Gupta and Peng to $(1+\varepsilon)$ in $\mathcal{O}(\sqrt{m}\varepsilon^{-2})$ [8]. The $\mathcal{O}(\sqrt{m})$ barrier was broken by Bernstein and Stein who gave a $(\frac{3}{2}+\varepsilon)$-approximation algorithm that achieves $\mathcal{O}(m^{1/4}\varepsilon^{-2.5})$ update time [2]. The same paper proposes an $(1+\varepsilon)$-approximation algorithm in very fast $\mathcal{O}(\alpha(\alpha+\log n)+\varepsilon^{-4}(\alpha+\log n)+\varepsilon^{-6})$ update time for the special case of bipartite graphs with constant arboricity. However, when allowing approximation in our model a much better results are possible. An $(1+\varepsilon)$ approximation in $\mathcal{O}(m\varepsilon^{-1})$ total time and with $\mathcal{O}(n\varepsilon^{-1})$ total length of paths was shown in [3].

3 Preliminaries

We consider the following matching problem. Let W and B be two sets of vertices over which the bipartite graph will be formed. The set W (called white vertices) is given up front to the algorithm, whereas the vertices in B (black vertices) arrive online. We denote by $G_t = \langle W \uplus B_t, E_t \rangle$ the bipartite graph after the t'th black vertex has arrived. The graph G_t is constructed online in the following manner. We start with $G_0 = \langle W \uplus B_0, E_0 \rangle = \langle W \uplus \emptyset, \emptyset \rangle$. In turn t a new vertex $b_t \in B$ together with all its incident edges $E(b_t)$ is revealed and G_t is defined as:

$$\begin{cases} E_t = E_{t-1} \cup E(b_t), \\ B_t = B_{t-1} \cup \{b_t\}. \end{cases}$$

The goal of our algorithm is to compute for each G_t the maximum size matching M_t. For simplicity we assume that we add in total $|W|$ black vertices. The final graph $G_{|W|}$ which is obtained in this process will be denoted by $G = (W \uplus B, E)$. We denote $n = |W| = |B|$ and $m = |E|$.

For every $t \in [n]$, we add orientation to edges of the graph G_t. This orientation is induced by matching M_t: the matched edges are oriented towards black vertices, while the unmatched edges are oriented towards white vertices. When a new vertex b_t arrives, we get an intermediate orientation $G^{\text{int}}{}_t = (E^{\text{int}}{}_t, B_t)$, where the edges of b_t are oriented towards its neighbors, and the rest of the edges is oriented according to M_{t-1}. Note that $G^{\text{int}}{}_t$ and G_{t-1} differ only by one vertex b_t. Any simple directed path in $G^{\text{int}}{}_t$ from b_t to some unmatched white vertex is an augmenting path. In turn t, if b_t can be matched, the edges of $G^{\text{int}}{}_t$ are reoriented along augmenting path π_t chosen by the algorithm, and the resulting orientation is G_t. The unmatched white vertices are called *seeds*. We denote the set of seeds after turn t as:

$$S_t = \{w \in W : wb \notin M_t \text{ for any } b \in B\}.$$

So in turn t the augmenting paths in $G^{\text{int}}{}_t$ are the directed paths from b_t to some $s \in S_{t-1}$. We refer to the seed of the path π_t from turn t as s_t, where $s_t \in S_{t-1}$. We represent a path as a graph consisting of path vertices and path edges. We use the notation $v \xrightarrow{\pi} v'$ to denote that a (directed) path π starts in v and ends in v', and $v \to v'$ to denote a connection via a directed edge. We use the notation $v \in \pi$ and $\rho \subseteq \pi$ to state that a vertex $v \in V(\pi)$ and that a path ρ is a subgraph of π, respectively. We also denote the length of a path π as $|\pi|$. Throughout the paper, when we write "at time t", what we formally mean is "in $G^{\text{int}}{}_t$".

The next thing we define is a set of vertices $\mathcal{D}_t$ called *dead* at time t. The set $\mathcal{D}_t$ is defined as the set of vertices in $G^{\text{int}}{}_t$ that cannot reach S_{t-1} via a directed path in $G^{\text{int}}{}_t$. Observe, that if at some point there is no directed path from a vertex to a seed, never again there will be such a path. If a vertex is dead, all vertices reachable from it are dead as well. Hence, no alternating path can enter such a dead region and reorient its edges to make some vertices alive. In other words, $\mathcal{D}_t \subseteq \mathcal{D}_{t+1}$ for every time moment t. Detailed matching-independent proof of this fact can be found in Section 1.2.2 of [11]. The vertices of $\mathcal{D}_t$ are called dead, while the remaining vertices are called alive.

We now define the effective degree of a black vertex b in turn t as the number of it's non-dead out-neighbors:

$$\text{degeff}_t(b) = |\Gamma_t(b) \setminus \mathcal{D}_t|$$

where $\Gamma_t(b)$ is the set of vertices v such that $b \to v$ in $G^{\text{int}}{}_t$, referred to sometimes as out-neighbours of b. In particular $\text{degeff}_t(b_t)$ is the number of all non-dead neighbors (in the undirected sense) of b_t, as all the edges adjacent to b_t are directed towards its neighbors.

Since we consider in this paper the special case when G_t is a tree at any time t, from now on we will refer to G as T, and to G_t as T_t.

4 Shortest Paths on Trees

In this section we study the shortest augmenting path (SAP) algorithm, which in each turn chooses the shortest among all available augmenting paths. We start by giving

an easy argument, that the total length of augmenting paths for SAP is $\mathcal{O}(n \log n)$ if all vertices b_t satisfy $\text{degeff}_t(b_t) > 1$. This shows that the difficult case is to deal with vertices of effective degree 1.

Lemma 1 *If for each $t \in [n]$ it holds that $\text{degeff}_t(b_t) > 1$, then the total length of all augmenting paths applied by* SAP *is bounded by $\mathcal{O}(n \log n)$.*

Proof Due to the definition of effective degree, every vertex b_t connects at least two trees T_1 and T_2 that contain a directed path connecting b_t with a seed. Let T_1 be a smaller of the two trees. The length of the shortest path π_t from b_t to a seed is at most the size of T_1. We charge the cost of π_t to $|\pi_t|$ arbitrary vertices of T_1. During the course of the SAP algorithm, every vertex can be charged at most $O(\log n)$ times, as each time it is charged, the size of its tree doubles. The total charge is hence $\mathcal{O}(n \log n)$. □

The main result of this paper and the subject of the remainder of this section is the bound for the general case, stated in the following theorem.

Theorem 1 *The total length of augmenting paths applied by* SAP *is $\mathcal{O}(n \log^2 n)$.*

In order to prove Theorem 1 we introduce a few definitions and observations. The core of our proof is the concept of a *dispatching vertex*.

Definition 1 A black vertex $b \in B$ is called dispatching at time t if b is the closest vertex to b_t on the path π_t that satisfies $\text{degeff}_t(b) > 1$. If there is no such vertex at time t we define s_t to be the dispatching vertex. We denote a dispatching vertex in time t as $\text{dis}(\pi_t)$. Moreover, for every dispatching black vertex b we define $\text{tlast}(b)$ as the moment when b is dispatching for the last time.

So every path π_t applied by SAP has a uniquely defined dispatching vertex $\text{dis}(\pi_t)$ assigned to it. The first observation we make is that we only have to care about suffixes of π_t's starting with $\text{dis}(\pi_t)$.

Definition 2 We split path π_t into two segments $\pi_t = \mu_t \rho_t$, where ρ_t is the suffix of π_t such that $\text{dis}(\pi_t) \xrightarrow{\rho_t} s_t$. Path $\mu_t = \pi_t \setminus \rho_t$ is the remaining part of π_t (a possibly empty prefix that ends in a vertex preceding $\text{dis}(\pi_t)$). We refer to the above defined suffixes as dispatching paths.

Lemma 2 *The total length of paths μ_t is linear in the size of the tree T, i.e.,*

$$\sum_{t \in [n]} |\mu_t| \in \mathcal{O}(n).$$

Proof The lemma holds due to Observation 2, proven below, which states that vertices of μ_t die at the time t when π_t is applied. With this observation it is clear that the time μ_t passes through a vertex is the last time SAP visits that vertex. So every vertex in the tree is visited by μ_t for any t at most once. □

Observation 2 *Vertices of μ_t die at the time t when π_t is applied.*

Proof At the time when π_t is applied, all vertices on μ_t have effective degree equal to 1, i.e., they have only one alive directed out-neighbour – their successor on μ_t. If we reverse the edges, the only chance for the vertices of μ_t to be alive is the last vertex b_t. This vertex, however, becomes dead because its only alive out-neighbour is removed. As a consequence the whole path dies. □

To bound the total length of augmenting paths π_t, it remains to bound the total length of dispatching paths: $\sum_{t\in[n]} |\rho_t|$. Consider the case when $\mathrm{dis}(\pi_t) = s_t$. Then the path ρ_t consists of a single vertex s_t. The total sum of paths ρ_t satisfying this case is thus $\mathcal{O}(n)$. It remains to consider the sum over all dispatching paths ρ_t that start in a black dispatching vertex. These non-trivial dispatching paths will be, from now on, the focus of our attention. In other words, our goal is to bound the following sum.

Lemma 3 *The total length of non-trivial dispatching paths is $\mathcal{O}(n \log^2 n)$:*

$$\sum_{\substack{t\in[n]:\\ \mathrm{dis}(\pi_t)\in B}} |\rho_t| \in \mathcal{O}(n \log^2 n).$$

For the sake of clarity, we split the proof of Lemma 3 into two steps, presented as Lemmas 5 and 6. More precisely, we partition the dispatching paths depending on whether $t < \mathrm{tlast}(\mathrm{dis}(\pi_t))$ or $t = \mathrm{tlast}(\mathrm{dis}(\pi_t))$, that is, if the dispatching path in question, is the last for its dispatching vertex (cf. Definition 1). In what follows, paths that satisfy the former condition are called *non-final* and their total length is bounded in Lemma 5, while *final* paths start at $b \in B$ when b is a dispatching vertex for the last time; their total length is the subject of Lemma 6.

These results will complete the proof of Theorem 1. However, before we jump into their proofs, we first briefly recall the heavy-light decomposition introduced by Sleator and Tarjan in [16] and state a related technical result, Lemma 4.

For a tree T rooted at r, the original technique partitions its edges into heavy and light, depending on whether the size of the subtree is strictly bigger than half of the size of the subtree rooted at parent. More precisely, let v be any vertex of T other than root r and set p_v to be its parent, then an edge $\{v, p_v\}$ is *heavy* if and only if $|subtree(v)| > \frac{1}{2}|subtree(p_v)|$, where $subtree(x)$ is a subtree of T rooted in $x \in V(T)$. Non-heavy edges are called *light*.

Observe, that because of the size requirements, each time we traverse a light edge away from the root r, the size of the current subtree halves. In other words, for any vertex v of T there are at most $\lfloor \log_2 |T| \rfloor$ light edges on the simple path from r to v. Note that each vertex can have at most two heavy incident edges, thus heavy edges form vertex-disjoint paths. Moreover, paths are of much simpler structure than arbitrary trees, hence allow for more efficient handling despite being possibly numerous.

For convenience, in this paper, we use a slightly modified version, that is, each non-leaf node selects exactly one *heavy* edge – the edge to the child that has the greatest number of descendants (breaking ties arbitrarily). In particular an edge may

be considered heavy, even if the subtree is strictly smaller than half of the size of the current tree. Just like in the original technique, the selected edges form the paths of the decomposition (each non-leaf vertex has at least one and at most two heavy edges), which we call *heavy paths*. By heavy-path(v) we denote the heavy path to which vertex v belongs, while level : $V(T) \to \mathbb{N}$ is the number of light edges on the simple path from a vertex to the root. Observe that level(v) $\leq \lfloor \log_2 |T| \rfloor$ for any vertex v of T.

Lemma 4 *Let T be any unrooted tree of size n. For any vertex v let $S^v = \langle S_0^v, S_1^v, \ldots \rangle$ be the sequence of subtrees of v (i.e., the connected components of $T \setminus \{v\}$) ordered descending by their size, that is, $|V(S_i^v)| \geq |V(S_{i+1}^v)|$. Then for:*

$$\Psi(v) = \sum_{i=2}^{|S^v|-1} |V(S_i^v)|,$$

we have $\sum_{v \in V(T)} \Psi(v) \in \mathcal{O}(n \log n)$.

Proof Let r be a centroid point of T, that is, a vertex such that $|V(S_0^r)| \leq \frac{1}{2}|V(T)|$. We root T at r, and perform the heavy-light decomposition of T. Observe that for all vertices $v \neq r$ we have that S_0^v contains r (it corresponds to the parent of v) and S_1^v corresponds to the biggest child of v. In other words, at most S_0^v and S_1^v can be connected by heavy edges, all the other subtrees $S_2^v, S_3^v, \ldots$ are connected by light edges.

Now we take an arbitrary vertex w and calculate how many times it can appear in $\sum_{v \in V(T)} \Psi(v)$. Suppose v is a vertex that counts w in $\Psi(v)$, then the first edge on the path from v to w has to be light. Moreover, S_0^v is not counted in $\Psi(v)$, so that path cannot pass through the parent of v. Because of that v has to be an ancestor of w. However, there are at most $\mathcal{O}(\log n)$ light edges on any path from w to the root r for any w. In other words, there can be at most $\mathcal{O}(\log n)$ vertices that count w in its sum of Ψ. Summing that for all vertices of T we get the desired bound of $\mathcal{O}(n \log n)$. □

With the help of Lemma 4 we can tackle the first part of Lemma 3, that is, the sum of the lengths of non-final dispatching paths.

Lemma 5 *The total length of non-final dispatching paths is $\mathcal{O}(n \log n)$:*

$$\sum_{\substack{t \in [n]: \\ b = \mathrm{dis}(\pi_t) \in B \\ t < \mathrm{tlast}(b)}} |\rho_t| \in \mathcal{O}(n \log n)$$

Proof Recall that the path π_t starts in the newly added vertex b_t. So in turn t either b_t is dispatching, or it dies. At any later time $t' > t$ at which b_t is dispatching $\pi_{t'}$ does not begin with b_t and hence one of b_t's neighbours dies based on Observation 2.

Consider a fixed vertex b and let $W_b^{\mathcal{D}} \subseteq W$ be the set of neighbors of b that die in turns when b is dispatching. The first time b is dispatching no neighbour of b dies, so $\mathcal{D}_t \cap W_b^{\mathcal{D}} = \emptyset$. The second time b is dispatching, it has at least one dead neighbour and set $\mathcal{D}_t \cap W_b^{\mathcal{D}}$ has exactly one element, namely the white vertex that preceded b on μ_t. More generally, the k-th time b is dispatching, $\mathcal{D}_t \cap W_b^{\mathcal{D}}$ has $k-1$ elements.

Suppose that the total number of times b is dispatching equals l, in particular we know that at some point of time $\mathcal{D}_t \cap W_b^{\mathcal{D}}$ will have $l-1$ elements. When b is dispatching for the k-th time set $\mathcal{D}_t \cap W_b^{\mathcal{D}}$ has only $k-1$ members. In other words, b has $l-k$ neighbors which are at that turn not yet in $\mathcal{D}_t \cap W_b^{\mathcal{D}}$, and thus alive. Furthermore b has at least two white neighbors that do not belong to $W_b^{\mathcal{D}}$ and are alive at the time when b is dispatching for the last time. Therefore, in total b has at least $l-k+2$ alive white out-neighbours.

We say that a subtree hangs from the neighbour w of b, if it is obtained by the removal of b from T and it contains w. Suppose that we discard two neighbors of b with the heaviest trees hanging from them, i.e., two heaviest neighbours. Then for $k = l-1$ we have at least one alive neighbor, for $k = l-2$ we have at least two alive neighbors, that is, at least one alive neighbor other than the neighbor used at $k = l-1$, and so on. In other words, for any $k < l$ we can find a distinct, not already assigned, alive neighbor w different than the two heaviest neighbors of b. However, the size of the subtree hanging from that neighbour bounds the length of the shortest augmenting path starting at b. Therefore, we can bound the total length of non-final paths dispatching at b by the total size of all subtrees of b except the two heaviest. Summing that up over the whole tree gives us a $\mathcal{O}(n \log n)$ upper bound, as shown by the previous lemma, Lemma 4. □

We now move on to the second part of Lemma 3, namely we bound the total length of final dispatching paths.

Lemma 6 *The sum of lengths of ρ_t such that $t = \mathrm{tlast}(\mathrm{dis}(\pi_t))$ is bounded from above by $\mathcal{O}(n \log^2 n)$:*

$$\sum_{\substack{t\in[n]:\\ b=\mathrm{dis}(\pi_t)\in B\\ t=\mathrm{tlast}(b)}} |\rho_t| \in \mathcal{O}(n \log^2 n).$$

To prove the above statement we will need a more fine-grained analysis than before. The problem with the shortest path approach is that its structure and the structure of matchings are very different. To close this gap we introduce yet another family of augmenting paths that relies much more on the structure of the tree. Obviously, because the shortest paths are shorter than any other path, any upper bound on the total length of the aforementioned new family of augmenting paths is an upper bound for the shortest paths as well.

Proof Similarly to Lemma 4, we root T at a centroid point and preform the heavy-light decomposition of the tree. That is, each vertex selects an edge to its largest subtree, which we call heavy, while all other edges are considered light.

We define λ_t as one of the M_{t-1}-augmenting paths that connect the newly-added vertex b_t to an unmatched white vertex. To be more precise, for each path λ connecting b_t with a free white vertex in turn t let the tuple

$$\langle \mathrm{level}(v_0),\ \mathrm{level}(v_1),\ \mathrm{level}(v_2),\ \ldots \rangle$$

represent λ, where $v_0, v_1, v_2, \ldots$ is the sequence of vertices of λ. We define λ_t as the path represented by the lexicographically last tuple. Observe that λ_t leaves any heavy path as soon as possible.

Now note that all M_{t-1}-augmenting paths starting in b_t are the same up to $\mathrm{dis}(\pi_t)$ and let $\mathrm{dis}(\lambda_t) = \mathrm{dis}(\pi_t)$. To bound the length of λ_t, we split it into three parts as follows:

$$\begin{aligned}\lambda_t' &= \mu_t,\\ \lambda_t'' &= (\lambda_t \setminus \lambda_t') \cap \text{heavy-path}(\mathrm{dis}(\lambda_t)),\\ \lambda_t''' &= (\lambda_t \setminus \lambda_t') \setminus \text{heavy-path}(\mathrm{dis}(\lambda_t)).\end{aligned}$$

Then λ_t'' follows heavy-path $(\mathrm{dis}(\lambda_t))$ at most up to the closest vertex b such that $\mathrm{tlast}(b) > t$, namely:

$$\left|\lambda_t''\right| \leq \min\left\{\mathrm{dist}(b, \mathrm{dis}(\lambda_t)) \mid b \in B\left(\text{heavy-path}(\mathrm{dis}(\lambda_t))\right), \mathrm{tlast}(b) > t\right\}. \quad (1)$$

The reason for this is that any vertex b with $\mathrm{tlast}(b) > t$ has at least three alive neighbors, with at least two of them reachable from b in $G^{\mathrm{int}}{}_t$, and at least one not on heavy-path $(\mathrm{dis}(\lambda_t))$. Any such b is a vertex at which λ_t'' can leave heavy-path$(\mathrm{dis}(\lambda_t))$.

Consider an arbitrary heavy path H and the set D of vertices on H that are ever dispatching in the entire run of the algorithm. Using $D_0 = D$ we split H into at least $d_0 = |D_0|$ non-empty fragments $h_0, h_1, \ldots, h_{d_0-1}$. Each such part $h \in \{h_0, h_1, \ldots, h_{d_0-1}\}$ has at least one of its endpoints in D_0, and usually both, unless h is the first or the last fragment. Thus, we can assign h to one of its ending vertices with preference for earlier turn tlast if there are two available. Formally $f_0 : \{h_0, h_1, \ldots, h_{d_0-1}\} \to D_0$, where:

$$f_0(h) = \operatorname*{arg\,min}_{b \in V(h) \cap D_0} \mathrm{tlast}(b).$$

Due to Inequality (1) in the previous paragraph we have:

$$\left|\lambda''_{\mathrm{tlast}(f_0(h_i))}\right| \leq |h_i| \text{ for any } 0 \leq i < d_0.$$

This means that the length of H bounds the total length of λ'''s related to the dispatching vertices in the image of f_0. However, as at most two h_i's can be assigned to the same vertex of D_0, the image of f_0 constitutes at least a half of D_0.

To take care of the rest of D_0, we iterate this reasoning. We construct a sequence of sets $D_0 \supseteq D_1 \supseteq \ldots$, each step halving the size of D_i. More precisely, we set:

$$D_i = D_{i-1} \setminus \left\{ f_{i-1}\left(h_j^{i-1}\right) \mid 0 \leq j < d_{i-1} \right\},$$

where $h_0^i, h_1^i, \ldots, h_{d_i-1}^i$ are the parts of H after the split by D_i and functions $f_i : \{h_0^i, h_1^i, \ldots, h_{d_i-1}^i\} \to D_i$ are defined as:

$$f_i(h^i) = \operatorname*{arg\,min}_{b \in V(h^i) \cap D_i} \mathrm{tlast}(b).$$

In other words, at most $\log n$ copies of H cover all λ'' paths related to H. Summing this up over all heavy paths gives us:

$$\sum_{\substack{t\in[n]:\\ b=\mathrm{dis}(\pi_t)\in B\\ t=\mathrm{tlast}(b)}} \left|\lambda''_t\right| \in \mathcal{O}(n\log n).$$

Furthermore, it also means that for any $v \in V(H)$ at most $\log n$ of λ''' paths may start in v. That is, $\log n$ copies of all non-heavy subtrees of $v \in V(H)$ cover all λ''' paths starting in v, which, by Lemma 4, implies

$$\sum_{\substack{t\in[n]:\\ b=\mathrm{dis}(\pi_t)\in B\\ t=\mathrm{tlast}(b)}} \left|\lambda'''_t\right| \in \mathcal{O}(n\log^2 n).$$

From the last two bounds we infer the statement of the Lemma 6. This also completes the proof of Theorem 1. □

5 Playing Against an Adversary

In the last section of this paper we discuss a quite surprising characteristic of Theorem 1 and its implications. Namely, nowhere in the proofs of Lemmas 3, 4, 5 and 6 we rely on the shape of any particular matching at any given turn, or even on the fact that these matchings are related to each other. To be more specific, we depend only on the structure of the tree, the properties of the dead and alive vertices, and the cardinality of the matchings. This leads us to a generalization of the setting in question and a respective counterpart of Theorem 1.

We define the *adversarial dynamic augmenting path setting* as a setup similar to the one from Section 3 in which, as before, each turn we are given a single black vertex with all its edges. However, the matching we use to calculate the shortest augmenting path is not the one produced by the algorithm in the previous turn, but some arbitrary matching of the same cardinality provided by the adversary. In particular, the edges might be oriented, wherever possible, away from the newly added vertex, thus making the augmenting paths the longest possible. Nonetheless, because we do not depend on the structure of the matching, the total length of all such augmenting paths is still small.

Corollary 1 *If the graph in the above setting is a tree, then the total length of all the shortest augmenting paths is* $\mathcal{O}(n\log^2 n)$.

It seems that this is true also in general bipartite graphs, and thus we form the following conjecture.

Conjecture 1 The total length of all the shortest augmenting paths in the setting above, that is, with the matching changing arbitrarily each turn, is still $\mathcal{O}(n\log n)$ worst case for any bipartite graph.

The ramifications of that conjecture are twofold. First, it suggests a new perspective and a new research angle in which we are allowed to change the matching to fit into some schema. That could possibly lengthen the paths in the process, but it might make the problem a bit more predictable and less dynamic, hence, in some aspects, easier. Second, it might allow for better algorithms. A matching procedure based on the above idea could alter the calculated matching during some turns in a random way, thus perhaps making its worst case less bad. As the reasons behind this phenomenon are far from clear, in authors' opinion Conjecture 1 is an interesting open problem.

References

1. Baswana, S., Gupta, M., Sandeep, S.: Fully dynamic maximal matching in $O(N)$ update time. In: Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS '11, pp. 383–392. IEEE Computer Society, Washington, DC (2011)
2. Bernstein, A., Stein, C.: Fully dynamic matching in bipartite graphs. 2015 to appear at ICALP (2015)
3. Bosek, B., Leniowski, D., Sankowski, P., Zych, A.: Online Bipartite Matching in Offline Time. In: 55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, pp. 384–393. IEEE Computer Society, Philadelphia (2014)
4. Chaudhuri, K., Daskalakis, C., Kleinberg, R.D., Lin, H.: Online bipartite perfect matching with augmentations. In: 28th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies INFOCOM 2009, pp. 1044–1052. IEEE, Rio De Janeiro (2009)
5. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. J. ACM **19**(2), 248–264 (1972)
6. Grove, E.F., Kao, M.-Y., Krishnan, P., Vitter, J.S.: Online perfect matching and mobile computing. In: Akl, S.G., Dehne, F., Sack, J.-R., Santoro, N. (eds.) Algorithms and Data Structures, volume 955 of Lecture Notes in Computer Science, pp. 194–205. Springer, Berlin (1995)
7. Gupta, A., Kumar, A., Stein, C.: Maintaining assignments online: matching, scheduling, and flows. In: Chekuri, C. (ed.) Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, pp. 468–479. SIAM, Portland (2014)
8. Gupta, M., Peng, R.: Fully dynamic $(1 + e)$-approximate matchings. In: 2013 IEEE 54th Annual Symposium on Foundations of Computer Science, vol. 0, pp. 548–557 (2013)
9. Ivković, Z., Lloyd, E.L.: Fully dynamic maintenance of vertex cover. In: Leeuwen, J. (ed.) Graph-Theoretic Concepts in Computer Science, volume 790 of Lecture Notes in Computer Science, pp. 99–111. Springer, Berlin (1994)
10. Karp, R.M., Upfal, E., Wigderson, A.: Constructing a perfect matching is in random nc. Combinatorica **6**(1), 35–48 (1986)
11. Leniowski, D.: On Maintaining Online Bipartite Matchings with Augmentations. PhD thesis, University of Warsaw (2015)
12. Mehta, A., Saberi, A., Vazirani, U.V., Vazirani, V.V.: Adwords and Generalized On-Line Matching. In: 46Th Annual IEEE Symposium on Foundations of Computer Science FOCS 2005, pp. 264–273 (2005)
13. Neiman, O., Solomon, S.: Simple deterministic algorithms for fully dynamic maximal matching. In: Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing, STOC '13, pp. 745–754. ACM, New York (2013)

14. Onak, K., Rubinfeld, R.: Property Testing. Chapter Dynamic Approximate Vertex Cover and Maximum Matching, pp. 341–345. Springer, Berlin (2010)
15. Sankowski, P.: Faster dynamic matchings and vertex connectivity. In: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 07, pp. 118-126. Society for Industrial and Applied Mathematics, Philadelphia (2007)
16. Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. J. Comput. Syst. Sci. **26**(3), 362–391 (1983)